Learning Time Series Patterns by Genetic Programming

Feng Xie **Andy Song** **Vic Ciesielski**

School of Computer Science and Information Technology
RMIT University
Melbourne,VIC 3001,Australia,
Email: feng.xie@rmit.edu.au

School of Computer Science and Information Technology
RMIT University
Melbourne,VIC 3001,Australia,
Email: andy.song@rmit.edu.au

School of Computer Science and Information Technology
RMIT University
Melbourne,VIC 3001,Australia,
Email: vic.ciesielski@rmit.edu.au

Abstract

Finding patterns such as increasing or decreasing trends, abrupt changes and periodically repeating sequences is a necessary task in many real world situations. We have shown how genetic programming can be used to detect increasingly complex patterns in time series data. Most classification methods require a hand-crafted feature extraction preprocessing step to accurately perform such tasks. In contrast, the evolved programs operate on the raw time series data. On the more difficult problems the evolved classifiers outperform the OneR, J48,Naive Bayes, IB1 and Adaboost classifiers by a large margin. Furthermore this method can handle noisy data. Our results suggest that the genetic programming approach could be used for detecting a wide range of patterns in time series data without extra processing or feature extraction.

Keywords: Genetic Programming, Pattern Recognition, Time Series

1 Introduction

Time series patterns describe how one variable or a group of variables change over a certain period of time. They are a critical factor in many real world problems where temporal information is essential. For example, any single point on an electrocardiogram is meaningless by itself, but a repeating sequence can reveal whether the rhythm of a heart is normal or abnormal. Similarly tasks like understanding patterns of climate change, recognizing words in audio waveforms and detecting target objects in videos all rely on finding patterns in the time domain. It is clear that a method which can capture regularities in temporal data is of great importance.

Handling temporal data is not a new area. Many methods have been proposed in the past. However they are significantly different as they were introduced for different tasks. For instance, temporal logic uses symbols and rules to represent time flow whereas a temporal database marks records with time stamps. Temporal databases introduced the concepts of transaction time and valid period for preserving the time information. Learning methods have been proposed for learning temporal rules or patterns as well. Sutton described the temporal difference(TD) learning method for prediction and proved its convergence and performance on a number of problems(1). Unlike traditional supervised machine learning approaches, the TD method learns from the differences between successive predictions to improve the final outcome. However this algorithm relies on the availability of an optimal set of features for the learning process. Boots and Gordon addressed this problem by adding a feature selection component to retain features which only contain predictive information(3). Neural networks have been used for handling temporal data as well. Dorffner summarized different types of artificial neural networks(ANNs) for this purpose(8). Furthermore ANNs has been combined with evolutionary strategies and a greedy randomised adaptive search procedure for time series forecasting(9).

These aforementioned existing methods do not constitute a generalized approach for learning time series patterns. Instead of operating on raw data, they often rely on some kind of additional processes to extract features from the time series data for learning, such as converting input signals from the time domain into the frequency domain, calculating variations at different points or transforming numeric data into symbolic data. The process for determining relevant features requires experience and domain knowledge from human experts. Manual interventions are usually essential here. Additionally the parameters and configurations are hand-crafted to suit only a certain problem. Therefore it is difficult to generalize a method for various problems. A generalized method for handling temporal data and learning underlying patterns still remain a big challenge.

In this study we propose genetic programming (GP) as a method for learning time series patterns without any extra processes such as data preprocessing or feature extraction. The aim of our investigation is to establish such a generalized method which can deal with different kind of scenarios such as detecting abrupt changes and recognizing various periodical patterns under one framework. Note the focus of this study is on recognizing patterns, not on time series prediction.

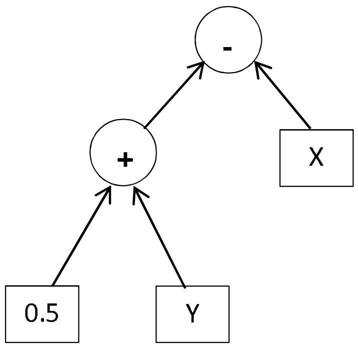

Figure 1: A Typical GP Program Tree $(y + 0.5) - x$

The main reason for using genetic programming is that GP has proved itself as a powerful and creative problem solving method. Even on some difficult problems such as image processing and computer vision, GP can work with unprocessed raw data to learn useful models. Essentially GP is performing two tasks here, extracting useful features and building a classification model accordingly. Typically these two tasks are treated as separate components in the conventional methods, but considered as one by GP-based method. This characteristic makes GP particularly suitable for problems of which the optimal features are unclear. Learning patterns in the time domain can be considered as such a problem. Because it is difficult to foresee what kind of features would be the most suitable if the nature of the underlying temporal patterns is not clear.

The rest of this paper is organized as such: Section 2 gives a brief background of Genetic Programming and its applications. Section 3 presents a collection of problems used in this study. Their difficulties increases gradually. The GP methodology and experiments are discussed in Section 4, while the discussions of our experiments are in Section 5. Section 6 concludes this study and discusses future investigations.

2 Background

Genetic Programming is a kind of evolutionary computation methods inspired by the survival-of-the-fittest principal. It was pioneered by Koza(2) who has successfully applied GP in many areas and even patented several solutions found by GP. A solution in GP is typically represented as a program tree on which the internal nodes are functions (operators) and the leaf nodes are terminals (operands). Initially a population of program trees are generated randomly as the first generation. These programs are evaluated on the problem to be solved. Based on the performance, each one of them is assigned with a fitness value. Solutions with higher fitness are more likely to be selected as the parents to generate a new population of solutions, namely the next generation. Programs in the new generation may be created by mutation (applying random change on a parent), crossover (exchanging tree branches between parents), or elitism (directly copying the best solutions from the previous generation). Figures 1 shows a simple GP tree.

There are existing studies using GP techniques to handle problems involving time series information. Kaboudan applied both neural networks and GP to the problem of forecasting housing prices based on both spatial and temporal information and suggested GP could produce more reliable and logically more acceptable forecasts(4). Song and Pinto(5) evolved programs to detect motion on live videos. GP was used to evolve programs to recognize interesting motions from background and uninteresting motions based on pixel values over a sequence of video frames.

Some researchers have investigated hybrid methods. Hetland and Sætrom presented a new algorithm combining GP and a pattern matching chip to discover temporal rules(6). The outcome of this experiment was comparable to some existing work. Another hybrid method is liquid state genetic programming proposed by Oltean(7). The core idea is dividing the whole system into two parts, the dynamic memory component and GP component. The former is kept by the liquid state machine while GP acts as a problem solver. These hybrid methods are computationally expensive because they require large memory and have long run times. It should be noted that the GP component here does not handle temporal rules directly.

As the aim of our work is to provide a GP-based method which does not require any extra components, we will take raw time series data as input and learn to recognize different patterns.

t_5	t_4	t_3	t_2	t_1	t_0	CLASS
1	0	0	0	0	0	Positive
1	1	0	0	0	0	Positive
0	0	0	1	1	1	Positive
0	0	0	0	0	1	Positive
0	0	0	0	0	0	Negative
1	1	1	1	1	1	Negative

Figure 2: Examples of Binary Pattern

3 Time Series Patterns

This section presents a collection of tasks investigated in this study. They are a group of artificial problems which are to represent tasks increasing level of difficulties: sequence of binary numbers, integers, floating point numbers, linear functions and periodic functions. Real-world applications will be studied in our future work.

3.1 Binary Patterns

This is the simplest problem where all the data points are either 0 or 1. One input sequence consists of six time-units worth of data, from t_0 the current value, to t_5 the value recorded 5 time-units ago. There are two types of patterns to be separated here. Negative means no change occurred during a period of six time-units. Positive means there is a change either from 0 to 1, or from 1 to 0 at any time within that period.

Note, in a positive sample, the point of change is not important, because detecting a change should not rely on a particular sampling position. For example, using one second as the time unit, a change occurring at 100th second should be captured by a 6-unit sampling window at multiple positions, from the 101st second to the 105th second. The direction of such change (either increase or decrease) is also not important. Multiple changes within one period such as 001100 are not considered here. Some examples are illustrated in Figure 2.

t_5	t_4	t_3	t_2	t_1	t_0	CLASS
5	5	6	6	6	6	Positive
7	100	100	100	100	100	Positive
1000	1000	1000	1000	100	100	Positive
6	6	6	6	6	6	Negative
100	100	100	100	100	100	Negative

Figure 3: Examples of Binary Pattern

t_1	t_0	CLASS
-12	-11.49	Positive
5.4	8.6	Positive
-5.63	-5.947	Negative
2233.2	2233	Negative

Figure 4: Examples of Floating Point Pattern (Threshold = 0.5)

3.2 Integer Patterns

The task here is very similar to the binary pattern, but the data points are integers with no restriction on the value. The length of a window is again 6 time-units. Any single change in values is considered as Positive while Negative means no changes. The total numbers of possible negatives and positives are enormous. Therefore a generalized rule to differentiate these two patterns is highly desirable. Examples are shown in Figures 3.

3.3 Floating-point Numbers with Threshold

The data points here are floating point numbers. Additionally a threshold is introduced. In real world applications, values which are close enough are often considered identical. Ignoring minor differences would be an advantage under this kind of circumstances. A hyper-sensitive detector would be equally bad as an insensitive one if not worse. Therefore data points with variations below a threshold are considered negatives. Otherwise they are positives.

Two types of tasks are studied. The simple version uses a window of length 2. Variations below 0.5 are considered no change. The other version uses a window of length 6, which is the same as the the one for binary and integer patterns. The threshold here is bigger as well, which is 5. Examples are shown in Figures 4 and 5.

3.4 Sine Waves and Random Numbers

In many real world scenarios no changes does not necessarily mean a constant value. Regular variations can be considered normal as well, for example the electric charge of alternating current. Under such circumstances, simply finding the existence of variation is not enough. Here a sine wave with an amplitude of 100 is used to generate negative samples, while positive samples are randomly generated numbers in the range of $[-100, 100]$.

t_5	t_4	t_3	t_2	t_1	t_0	CLASS
7	1000	239	1000	43.9372	1000	Positive
4	9.21	4.3	6.23	5.4	7.32	Positive
1	-0.3	0.94	2.953	0.32	2.04	Negative
2232.2	2233	2231	2232	2231.3	2333	Negative

Figure 5: Examples of Floating Point Pattern (Threshold = 5)

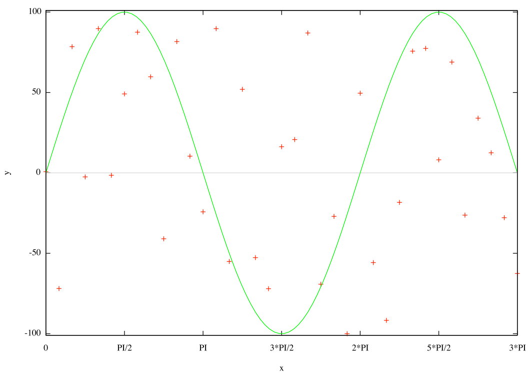

Figure 6: A Sine Wave: $y = 100 \times sin(x)$ and A Random Sequence

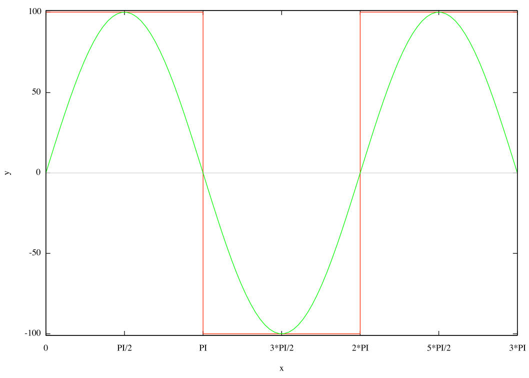

Figure 7: A Sine Wave: $y = 100 \times sin(x)$ and A Step Function

These data points can be visualized in Figure 6. For sine waves, values are taken at intervals of 15 degrees. To enable the learning process to capture the characteristics of a sine wave, 3π e.g. one and a half periods of data are included. This means that each sample contains 37 consecutive points sampled along the time line. This is much bigger than that in the previous tasks.

Note that only one sine wave is shown in Figure 6. Sine waves could start from different phases. Therefore negative samples consist of a collection of sine waves with 15 degree shift. So all the negative samples are different. A good model should consider them as the same class, but report random sequences as anomalies.

3.5 Sine Waves and Other Periodical Functions

The previous task might be not challenging enough as the random sequence has no regularities at all while the sine waves do. This could provide hints for a learning process. So other periodic functions are introduced as positives here. They are shown in Figures 7 and 8. The first is a step function which has oscillating values from 100 to -100. The second is a triangle wave of which the value varies from 100 to -100 as well. Furthermore all these functions have an identical frequency. Samples of both negatives and positives are taken with 15 degree shifts. Hence all samples for the step function and the triangle function are different.

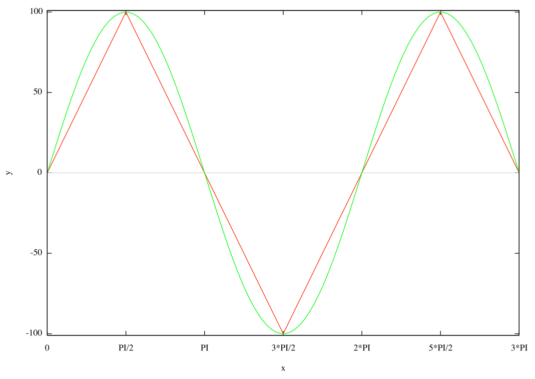

Figure 8: A Sine Wave: $y = 100 \times sin(x)$ and A Triangle Function

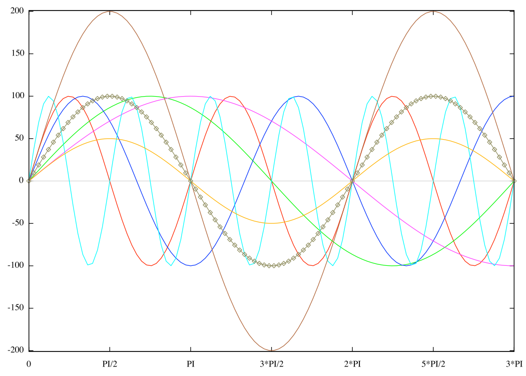

Figure 9: Target Sine Wave: $y = 100 \times sin(x)$ (in dotted curve) and Other Sine Waves

3.6 Different Sine Waves

To make the task even more challenging, different sine waves are mixed in this dataset. As shown in Figure 9, one particular sine wave is set as the target while other Sine waves with different frequencies and amplitudes are marked as negatives.

3.7 Patterns with Noise

Often signals in application would contain noise. To investigate how noisy data are handled by different learning methods, we add random noise to the patterns described in Section 3.5 and Section 3.6. The range of noise is in between $[-1, 1]$.

4 Methodology and Experiments

The GP method is briefly described in this section. Table 1 shows the function set we used. In addition to basic arithmetic operators, conditional operators are included to perform value comparison. The terminal set simply contains the input variables and random constants.

Table 3 shows the runtime parameters of our experiments. One objective of this study is to obtain solutions which are human comprehensible, so we could understand the learned models. Therefore a relatively small tree depth is used. Furthermore the population size is rather small because we aim to use as few evaluation as possible.

For comparison purposes a number of classical classifiers were used for all the tasks described above. OneR is the simplest classifier which builds rules based on one attribute(10). IBk is an instance based

Table 1: GP Function Set

Function	Return Type	Arguments
$+$	Double	Double,Double
-	Double	Double,Double
$\times$	Double	Double,Double
/	Double	Double,Double
if	Double	Boolean,Double,Double
$>$	Boolean	Double,Double
$<$	Boolean	Double,Double

Table 3: GP Runtime Parameters

Maximum Depth of Program	10
Minimum Depth of Program	2
Number of Generations	100
Population Size	10
Mutation Rate	5%
Crossover Rate	85%
Elitism Rate	10%

algorithm which classifies the target according to its closest neighbour in feature space(13). The k value is 1 in all experiments. NaiveBayes is a probability based classification method(12). J48 generates decision trees based on the information gain of each attribute(11). Instead of using one classifier, multiple classifiers could be combined as an ensemble to improve the performance. Therefore AdaBoost was also used (14). For each task, the best performer, either OneR, or J48, or NaiveBayes or IB1, is selected as the base classifier in AdaBoost.

For each task the same set of examples are supplied to GP and to other methods for training and test. All data sets include both positive and negative cases. The number of both cases for each task is listed in Table 2. Two thirds of data were for training and one third for test. Table 4 lists the test accuracies achieved by all these methods on various tasks, numbered from No.1 to No.11. Each row in the table represents the results obtained by different methods for one particular task . GP solutions were evolved at least ten times. The test accuracies under GP are results from the best individuals.

As shown in Table 4, GP consistently outperformed the classical methods. There were only two cases that these methods could match GP: AdaBoost for binary patterns and instance-based learning (IB1) for differentiating sine wave and sequences of random numbers. All these methods performed poorly on handling floating-point numbers especially when there are 6 consecutive values (No. 4), while GP still achieved reasonably high accuracy. These methods also performed poorly on another rather difficult task, distinguishing different sine waves (No. 8) while GP achieved 100% accuracy. Even after adding noise to patterns, GP was still able to achieve better results compared to these classical classifiers.

5 Discussion

Most of the GP runs terminated around the 30th to 50th generations because a perfect solution was found. This suggests that the representation described earlier is appropriate for recognizing these patterns, so solutions could be found quickly.

To understand the behavior of evolved programs we examined some of the best individuals. Although most of these programs are not quite comprehensible, such analysis does provide some insights. For the simple version of floating-point numbers (No.3, 2 units,

Table 2: Number of Positive and Negative Instances for Each Task

		Positive Instances	Negative Instances
1.	Binary Pattern	10	2
2.	Integer Pattern	52	10
3.	Floating-Point (2 Units)	37	25
4.	Floating-Point (6 Units)	37	25
5.	Sine Wave vs. Random Numbers	101	24
6.	Sine Wave vs. Step Function	127	24
7.	Sine Wave vs. Triangle Wave	144	24
8.	Different Sine Waves	134	24
9.	Sine Wave vs. Step Function(With Noise)	127	24
10.	Sine Wave vs. Triangle Wave(With Noise)	144	24
11.	Different Sine Waves(With Noise)	134	24

Table 4: Test Accuracies in Percentages(%)

		OneR	J48	NBayes	IB1	AdaBoost	**GP**
1.	Binary Pattern	83.3	83.3	83.3	83.3	100	100
2.	Integer Pattern	85.71	85.71	85.71	90.48	90.48	100
3.	Floating-Point (2 Units)	76.19	61.9	57.14	66.67	76.19	100
4.	Floating-Point (6 Units)	69.23	61.54	60.97	53.85	53.85	92.68
5.	Sine Wave vs. Random Numbers	86.05	79.07	81.4	100	95.35	100
6.	Sine Wave vs. Step Function	88.68	88.68	50.94	92.45	92.45	100
7.	Sine Wave vs. Triangle Wave	86.2	81.03	56.9	89.66	89.66	100
8.	Different Sine Waves	11.76	41.18	43.53	78.82	82.35	100
9.	Sine Wave vs. Step Function(With Noise)	52.83	88.68	50.94	92.45	92.45	98.11
10.	Sine Wave vs. Triangle Wave(With Noise)	79.31	87.93	56.9	89.66	89.66	94.34
11.	Different Sine Waves(With Noise)	17.65	40	43.53	78.82	82.35	92.94

threshold 0.5), one program behaves like this:

$$(t_1 - t_0 < 1)?Negative : Positive \qquad (1)$$

The decision is simply based on the difference between the two values. For separating sine waves and random numbers (No. 5), one best program is effectively

$$(t_4 + t_{16} == 0)?Negative : Positive \qquad (2)$$

The distance between t_4 and t_{16} is exactly π, half of the period. Therefore the sum of these two values on a sine wave should be always zero regardless the phase of the wave. This suggests that the evolved GP program did capture a defining characteristic of the periodic function.

One might argue that given appropriate features such as calculating difference between consecutive points, finding the frequency by Fourier transform and so on, the other methods could perform equally well. Certainly such processes would be helpful for learning. However as discussed before, such a process requires domain knowledge from human experts who understand the problem itself. Automatically generating optimal features for the task in hand is often difficult. Additionally it can not be generalized for other problems. There is no universal feature set which is suitable for all kinds of patterns. GP combines the feature finding and classification process together.

6 Conclusion and Future Work

A Genetic Programming based method is presented in this study for learning time series patterns. Eleven groups of patterns with increasing difficulties were used to evaluate this GP method. In comparison with five well known machine learning methods: a rule based classifier, a decision tree classifier, a Naive Bayes classifier, an instance based classifier and Ada Boosting, the evolved programs achieved perfect accuracies on most of the tasks and consistently outperformed the other classifiers. We conclude that the presented GP method is suitable for learning time series patterns. This method has clear advantages, as it is capable of finding characteristics directly on raw input to differentiate various patterns rather than on manually defined features. No extra process is required by this method. Additionally it is capable of handling noisy data input.

Our future work will go beyond a single variable because in many scenarios such as climate change and video analysis, one must be able to handle multiple variables which may or may not be independent to each other. Another extension is treating monotonicity as a pattern, so a "normal" variable should always be stable or change in one direction and never oscillate. Mixtures of multiple patterns is another area to explore, such as a step function on top of a sine wave. This method will be applied on real world applications in the near future.

References

[1] Sutton, R.S.: Learning to predict by the methods of temporal differences, Machine learning, vol. 3, pp.9–44 (1988)

[2] Koza, J.R.: Genetic programming I:On the programming of computers by means of natural selection, vol. 1, MIT press(1996)

[3] Byron B. and Geoffrey J. G.: Predictive State Temporal Difference Learning, Proceedings of Advances in Neural Information Processing Systems 24(2010)

[4] M. A. Kaboudan: Spatiotemporal Forcasting of Housing Price By Use of Genetic Programming, the 16th Annual Meeting of the Association of Global Business,2004

[5] Andy S. and Brian P.: Study of GP representations for motion detection with unstable back-

ground, IEEE Congress on Evolutionary Computation (2010)

[6] Magnus L. H. and Pal S.: Temporal Rule Discovery using Genetic Programming and Specialized Hardware, Proceedings of the 4th International Conference on Recent Advances in Soft Computing (2002)

[7] Oltean, Mihai:Liquid State Genetic Programming, Proceedings of the 8th international conference on Adaptive and Natural Computing Algorithms (2007)

[8] Georg D.:Neural Networks for Time Series Processing, vol. 6, pp.447-468, Neural Network World, 1996

[9] Aranildo R. L. J.and Tiago A. E. F.: A hybrid method for tuning neural network for time series forecasting, pp.531–532, Proceedings of the 10th annual conference on Genetic and evolutionary computation, 2008

[10] R.C. Holte: Very simple classification rules perform well on most commonly used datasets,vol. 11, pp.63–91, Machine Learning, 1993

[11] Ross Quinlan: C4.5: Programs for Machine Learning, Morgan Kaufmann Publishers, 1993

[12] George H. J, Pat L.: Estimating Continuous Distributions in Bayesian Classifiers, Eleventh Conference on Uncertainty in Artificial Intelligence, pp.338–345, 1995

[13] D. Aha, D. Kibler: Instance-based learning algorithms, vol. 6, pp.37–66, Machine Learning 1991

[14] Yoav Freund, Robert E. S.: Experiments with a new boosting algorithm, pp.148–156,Thirteenth International Conference on Machine Learning,1996